CHAPTER 1

What Is UML?

✦ ✦ ✦ ✦

In This Chapter

Defining UML

Exploring the history of the standard

Understanding the motivation behind the UML standard

Examining the fundamental concepts of UML

Exploring the goals of UML

Examining the scope and features of the UML

Understanding the role of the Object Management Group (OMG)

Understanding how UML fits into the Model-Driven Architecture (MDA)

✦ ✦ ✦ ✦

The Unified Modeling Language (UML) has been formally under development since 1994. UML is a distillation of three major notations and a number of modeling techniques drawn from widely diverse methodologies that have been in practice over the previous two decades. During this time it has had an undeniable impact on the way we view systems development. Despite early competition from existing modeling notations, UML has become the de facto standard for modeling object-oriented software for nearly 70 percent of IT shops. UML has been adopted by companies throughout the world, and today more than 50 commercial and academic modeling tools support software and business modeling using UML.

UML enables system developers to specify, visualize, and document models in a manner that supports scalability, security, and robust execution. Because UML modeling raises the level of abstraction throughout the analysis and design process, it is easier to identify patterns of behavior and thus define opportunities for refactoring and reuse. Consequently, UML modeling facilitates the creation of modular designs resulting in components and component libraries that expedite development and help insure consistency across systems and implementations.

Unlike previous methodologies, you don't have to change the way you work just to suit the demands of a vendor or methodologist. UML uses extension mechanisms to customize UML models to a particular application type or technology. While the extension mechanisms are a bit limited today, they do provide substantial support for tailoring UML to the needs of a specific project, whether the project's goal is a transaction-oriented application, real-time or fault-tolerant system, or e-commerce or Web service, and regardless of the subject domain.

UML profiles collect predefined sets of extension mechanisms for a specific environment. For example, in the UML specification itself you will find profiles for J2EE, COM, .NET, and CCM development. Each profile provides customized modeling elements that map to the common elements and features in each of these architectures. This approach enables the modeler to focus time and energy on the project content instead of the unique modeling features of the implementation domain.

The standardized architecture of UML is based on the Meta-Object Facility (MOF). The MOF defines the foundation for creating modeling languages used for object modeling, such as UML, and for data modeling, such as the Common Warehouse Model (CWM). The MOF defines standard formats for the key elements of a model so that they can be stored in a common repository and exchanged between modeling tools and languages. XML Metadata Interchange (XMI) provides the mechanism to implement the sharing of these modeling elements between modeling tool vendors and between repositories. This means, for example, that a project can use one tool for developing a platform-independent model (PIM) using UML diagrams and switch to another tool to refine the model into a platform-specific model (PSM) using a CWM model to generate the database schemas. This standards-based approach places the choice of tools in the hands of the modelers instead of the tool vendors.

UML models can be precise enough to generate code or even the entire application. Automated test suites can verify the accuracy of the model. When coupled with tools to compile the UML model, the model can even be executed before any code exists. Vendors (for example, Kabira Technologies `www.kabira.com` and Project Technology, Inc. `www.projtech.com`) are already providing compilers that are being used in projects today. A fully executable UML model may be deployed to multiple platforms that each use different technologies. A model might be deployed in one place using one language, middleware, and database configuration, and at another location with an entirely different configuration. The mapping of the model to an implementation configuration is accomplished using a profile, with a separate layer that maps the two requirements, the model and the implementation environment. To use the model in other implementation environments, simply create a new profile. Thus the UML profile represents a *level of indirection* between the model and the implementation environment, freeing each to be created independently of the other.

All of these features didn't appear overnight. A great deal of collaborative effort was invested to create the current standard, and not without conflict. You may have already heard people taking sides on a variety of modeling issues and practices that arise when they try to use UML. To clarify some of the reasons behind these debates, I begin with a brief history explaining how UML first came to be. If you can understand the process behind the ongoing development of the standard, you will be better equipped to follow the changes between version 1.4 and the new developments in version 2.0 described throughout the rest of this book. Even more important, you need to understand how UML fits into the much larger plan by the Object Management Group (OMG) to standardize systems development with Model-Driven Architecture (MDA).

Since UML 2.0 is pending as of the writing of this book, I have included both UML 1.4.1 and UML 2.0. I hope this will help those of you who might be using modeling tools based on UML 1.4.1 until you are able to upgrade. At the same time it should give you some insights for evaluating either your existing vendor's implementation of UML 2.0 or other new modeling products. For a more complete explanation of this approach, refer to "How to read this book" in the Preface.

Note When the OMG added Action Semantics to the 1.4 specification, it originally called it "UML 1.4 with Action Semantics." It later appeared on the OMG site as UML 1.5. So don't be surprised if I accidentally bounce between references to 1.4 and 1.5.

The rest of this chapter discusses

✦ The history of the UML through version 1.4

✦ The goals, scope, and features of the UML

✦ The objectives of UML 2.0

✦ The role of the Object Management Group (OMG)

✦ How UML fits into the bigger picture: The OMG's Model-Driven Architecture (MDA) initiative

Understanding the History Behind UML

UML is designed specifically to represent object-oriented (OO) systems. Object-oriented development techniques describe software as a set of cooperating blocks of information and behavior. For example, a performance at a theater would be coded as a discrete module with its own data about dates and time, and behavior such as schedule, cancel, or reschedule all rolled together. This was a stark departure from the old notion that data resides in files and behavior lives in programs.

The effect of this simple idea, the combining of data and behavior into objects, had a profound effect on application design. As early as the 1970s, a number of methods were developed to exploit the new object-oriented (OO) programming concepts. Developers quickly recognized that object orientation made possible a development process in which the way that they talk about the application corresponds directly to how they code it. They also found that it was relatively easy to draw (model) the objects so that they could talk about the design. Each object was represented as an element on a diagram. Because the model elements were almost identical to the code elements, the transition from model to code was simple and efficient. Moving design discussions up to the models instead of the code helped the developers deal with design issues at a high level of abstraction without getting caught up in the coding syntax.

03 526049 Ch01.qxd 8/20/03 11:47 PM Page 6

Early Modeling Methodologies

Software developers weren't the only people who discovered the benefit of modeling. Other engineering disciplines such as database management and design were also creating modeling techniques such as Entity Relationship modeling (ER diagrams) and Specification and Description Language (SDL). It quickly became clear that modeling provided a way to cope with complexity, encourage collaboration, and generally improve design in all aspects of software development.

The need for modeling solutions increased with the growth in numbers and sophistication of software systems. Systems were growing rapidly in complexity and required more and more collaboration and solid, durable design quality. Modeling had proven itself in exactly these circumstances. Literally hundreds of people sprang to work developing modeling methodologies to solve the growing problem. But the resulting proliferation of solutions caused some problems. The widely diverse efforts were inefficient in that they lacked the necessary collaboration to produce results that could be widely applied by the IT community. In fact, the diversified approach resulted in what were affectionately called the "method wars," battles between method authors with their loyal followers pitted against one another over who had the best solution. Authors of each method vied for support for their methods. Tool vendors labored to support many different notations in the same tool. Companies struggled to identify and follow a single "best" method, train their people, and support the method only to find that no one method could fully meet their needs.

The proliferation of isolated solutions and the associated battles were signals that the need for a comprehensive solution for software modeling was a priority. The solution needed to be flexible, scalable, secure, and robust enough to handle the diverse software and business environments of the present and the future.

The Creation of UML

By the early 1990s, a few leaders had emerged from the field of methods and notations. Object-Oriented Software Engineering (OOSE), developed by Ivar Jacobson, is based around the use-case concept that proved itself by achieving high levels of reuse by facilitating communication between projects and users, a key success factor for IT projects. James Rumbaugh developed the Object-Modeling Technique (OMT) with an emphasis on the analysis of business and data intensive systems for defining a target problem, a second key success factor for IT projects. The Booch method, developed by Grady Booch, had particular strengths in design and implementation, defining and mapping a solution to the target problem, a third key to successful IT projects. These significant contributions are like the legs on a three-legged stool: the combination of the three methods and their notations supported the entire range of requirements needed to create a single, comprehensive software-modeling standard.

It is important to point out that many other methods provided some of the same three key factors. The difference is that they did not aggressively seek to combine their efforts to address the bigger picture, a standards-based approach to modeling software. In October 1994, Grady Booch and Jim Rumbaugh, working at Rational Software Corp., started merging their two methods. The independent evolution of their two products was bringing the methods closer together anyway; Booch was adopting more of an analysis focus and Rumbaugh was assuming more of a design focus. Now the deliberate reconciliation began in earnest. The effort resulted in a greatly simplified notation and a deliberate effort to address the need for a true *language architecture* rather than simply a notation. An architectural approach would bring the needed semantic integrity and consistency for a durable standard.

A year later, in the fall of 1995, Booch and Rumbaugh had completed the first draft of the merged method referred to as Unified Modeling Language version 0.8. About the time that the draft was completed, Ivar Jacobson and his company, called Objectory, joined Rational Software Corp., and the "three amigos"—Booch, Rumbaugh, and Jacobson—began integrating OOSE into the UML standard. The use-case concept brought to UML the essential user-centric elements that completed the range of features to make UML the comprehensive standard that it needed to be to gain wide acceptance.

Booch, Rumbaugh, and Jacobson established four goals for the Unified Modeling Language:

1. Enable the modeling of systems (not just software) using object-oriented concepts
2. Establish an explicit coupling to conceptual as well as executable artifacts
3. Address the issues of scale inherent in complex, mission-critical systems
4. Create a modeling language usable by both humans and machines *(UML 1.4, pgs. 1-12,13)*

The result of the collaborative effort of the three amigos was the release of UML versions 0.9 and 0.9.1 in the fall of 1996. However, despite the fact that they sought feedback from the development community, they recognized the need for broader involvement if the UML was truly to be a standard.

Enter the Object Management Group (OMG), the standards body that brought us CORBA, Interface Definition Language (IDL), and the CORBA Internet Inter-ORB Protocol (IIOP). By this time UML was being recognized as vital to the goals of many companies. It was in their best interest to see that the standard get the support it needed to be completed. In response to this overwhelming need, the OMG published a Request for Proposal (RFP), and then the Rational Software Corporation created the UML Partners consortium, which was committed to finishing what the three amigos had started. Contributing members of the consortium included a mix of vendors and system integrators: Digital Equipment Corporation, HP, i-Logix, IntelliCorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational Software, TI, and Unisys. The result of their efforts was published in January 1997 as UML 1.0.

At the same time, another group of companies (IBM & ObjecTime, Platinum Technologies, Ptech, Taskon & Reich Technologies, and Softteam) was working on and submitted another proposal for UML. And, exemplary of the UML history, the alternative proposal was viewed not as competitive, but collaborative. The new team joined the UML Partners consortium and the work of the two groups was merged to produce UML 1.1 in September 1997. Since then, the OMG has assumed formal responsibility for the ongoing development of the standard, but most of the original consortium members still participate.

In reading this brief history you've probably noticed that this all happened pretty fast. The drive to deliver the final version so quickly had its consequences. While the architecture infrastructure and even the superstructure were relatively well defined, some problems remained. For example, the Activity diagram did not have the ties to the state machine semantics required to support all of the features and notations needed for real business modeling. Also, many of the Standard Elements were added hastily and had not been fully defined. Most important, the meta-modeling approach fell short of the desired implementation, making it difficult to align UML with the Meta-Object Facility (MOF), a foundation technology in the OMG's MDA strategy. Fortunately, the standard is still evolving.

The OMG set up a Revision Task Force (RTF) to oversee the ongoing evolution of the UML standard. The RTF is responsible for addressing all questions, changes, and enhancements to UML and for publishing subsequent releases. To date, the RTF has taken up more than 500 formal usage and implementation issues submitted to the OMG for consideration. In fact, you can submit your own suggestions and comments on existing issues to `uml-rtf@omg.org`.

The standard has since progressed through version 1.3 (1.2 was a purely editorial revision) and on to version 1.4. The most recently adopted specification (September 2002) is version 1.4.1 with Action Semantics, which, as the name implies, added action semantics to the 1.4 specification. Action Semantics is a critical element in the creation of executable UML models.

Cross-Reference

To learn more about Action Semantics, refer to Chapter 19.

The Goals and Features of UML

UML is designed to meet some very specific objectives so that it can truly be a standard that addresses the practical needs of the software development community. Any effort to be all things to all people is doomed to fail, so the UML authors have taken care to establish clear boundaries for the features of the UML.

The next section explains the objectives and the scope of the UML, the fundamental features provided by the UML, and discusses the role of the OMG in the management and ongoing development of the UML as a part of its MDA strategy.

The goals of UML

The OMG knows that the success of UML hinges on its ability to address the widely diverse real-world needs of software developers. The standard will fail if it is too rigid or too relaxed, too narrow in scope or too all-encompassing, too bound to a particular technology or so vague that it cannot be applied to real technologies. To ensure that the standard will, in fact, be both practical and durable, the OMG established a list of goals.

UML will

- ✦ Provide modelers with a ready-to-use, expressive, and visual modeling language to develop and exchange meaningful models.
- ✦ Furnish extensibility and specialization mechanisms to extend the core concepts.
- ✦ Support specifications that are independent of particular programming languages and development processes.
- ✦ Provide a formal basis for understanding the modeling language.
- ✦ Encourage the growth of the object tools market.
- ✦ Support higher-level development concepts such as components, collaborations, frameworks, and patterns. (UML1.4 specifications)

Each of these goals is discussed in detail in the following sections.

Goal 1: Provide modelers with a ready-to-use, expressive, and visual modeling language to develop and exchange meaningful models

UML must be defined at a level that allows it to be used as-is off the shelf. Modelers should be able to start building diagrams without first customizing the notation to their development environment, programming language, or application. The modeling language should work equally well for Java and C++, for accounting and aviation.

To accomplish this, the standard has to define the semantics of the modeling language as well as the visual representation of the language. Semantics provide the rigor that ensures the consistent application of the models and model elements. A consistent visual representation of the model elements facilitates adoption and use of the modeling technique.

The standard also must be comprehensive but not exhaustive. It must include all the core modeling elements common to most, not all, software projects. If it is not complete, modelers will not be able to use it without customization. If it is exhaustive — well, it just can't be. Instead, the OMG adopted the second goal.

Goal 2: Furnish extensibility and specialization mechanisms to extend the core concepts

In overly simplified terms, the core concepts should represent the old 80/20 rule. We should be able to build 80 percent of the systems out there with 20 percent of the conceivable concepts. When these core concepts are not enough, there should be a way to build on them to get what we need.

Wherever possible a modeler should not have to invent entirely new concepts. Users should be able to use concepts already defined by UML. There are at least three ways that UML enables modelers to create new model elements:

✦ The core defines a number of fundamental concepts that may be combined to create the new concept.

✦ The core provides multiple definitions for a concept.

✦ UML supports the ability to customize a concept by specializing one or more of its definitions. (To specialize means to use an existing definition and then override and/or add elements to it.)

A UML-defined solution for wholesale extensibility is a profile. A *profile* is basically an implementation of UML for a specific domain, such as a particular technology platform or a specific line of business. A profile predefines a set of model elements that are unique or simply common to the target environment. In this manner, profiles tailor the modeling elements so that the modeler can represent his environment more accurately than is possible with generic UML but without losing any of the semantic clarity of UML concepts.

Goal 3: Support specifications that are independent of particular programming languages and development processes

One very valuable reason for modeling is to separate the requirements from the implementation. Tying UML to a particular language automatically alienates everyone not using that language. An implementation also ties UML to a point in time. For example, when the programming language changes, UML becomes obsolete until it can be brought up to date.

However, UML must map to the common object-oriented design constructs defined in most OO languages. This alignment will support code generation and reverse engineering, the integration of the modeling and coding environments. But rather than alter UML to conform to languages, the mapping is accomplished through profiles that define the relationships between the model elements and the implementation constructs. Using a separate mapping layer effectively decouples, or separates, UML from the implementation languages, allowing both to evolve at their own pace.

03 526049 Ch01.qxd 8/20/03 11:47 PM Page 11

Goal 4: Provide a formal basis for understanding the modeling language

The language must be defined at a level that is precise yet accessible. Without precision, the models do not help define a real solution. Without accessibility, no one will use it. The UML standard uses Class diagrams to represent the formal definitions of objects and their relationships. Each Class diagram is supplemented with text detailing the semantics and the notation options. The constraints that define the integrity of the model elements are expressed using Object Constraint Language (OCL). (See Chapter 18.)

Goal 5: Encourage the growth of the object tools market

The modeling tool market is dependent on a unified standard for modeling, for the model repository, and for model interchange. To the extent that vendors can rely on a stable standard, they can quickly and effectively implement all three fundamental tool features. As the vendors' cost to provide the core functionality decreases, vendors are freed to pursue value-added modeling-environment enhancements such as integration with coding environments, database management tools, syntax checking, model verification, and more.

We are seeing the effect of the standard today. The number of tools has mushroomed, and the feature sets offered in the tools have exploded. Where tools used to focus almost exclusively on just being able to draw diagrams, today they are performing syntax checking of OCL statements, diagram synchronization, code generation and reverse engineering, importing from various other tools, exporting HTML or XML reports, supporting integration with one or more coding environments, and much more.

Goal 6: Support higher-level development concepts such as components, collaborations, frameworks, and patterns

The standard needs to support the modeling of higher-level concepts such as frameworks, patterns, and collaborations. Doing so supports the advancement of modeling and systems development. By ensuring this future potential, UML becomes an asset that facilitates technological evolution rather than being one more legacy that has to be dragged into the future with all the other old technologies.

The scope of the UML

UML is designed to be the merging of best development practices and the leading modeling concepts of the past 30 years. There is also a deliberate effort to take into account the fact that development technologies and techniques are always changing.

With such an ambitious goal, it would be easy to fall into the trap of making UML define everything — modeling, development methodology, project management, systems integration, and so forth — about the software development process. So the first and most visible boundary established by the OMG was to define only the modeling language, including the semantics and the notation for creating models. Therefore, UML defines only the modeling elements used to describe the artifacts of software development. It does not describe any process for creating those artifacts. In fact, the intent of the standard is to create a language that may be used with any process, much like you could hand someone a hammer and say, “Hang this picture” or “Build a house.” The same tool may be used for very different tasks. In the same manner, UML might be (and is) used with the Rational Unified Process, Shlaer/Mellor, Agile Modeling, or any number of proprietary methodologies.

UML also says nothing about programming languages. The object-oriented concepts applied in modeling are the same concepts applied in OO programming languages, but the relationship begins and ends with this common foundation. For example, Java does not support multiple inheritance but UML does. Neither Java nor UML is going to change because of this inconsistency. They each have their own goals and audiences that drive their choices regarding how to support OO concepts. Again, the UML standard cannot be tied to a particular technology without losing its ability to keep pace with advancements in technology.

Finally, UML does not seek to usurp other modeling techniques such as Business Process Re-engineering (BPR) flowcharts or entity-relationship modeling. However, it has proven itself to be robust enough to bring added precision, comprehensiveness, and flexibility to the same modeling domains. The infrastructure of UML (covered later in this chapter) is actually designed to be the basis for defining any number of modeling languages. In the event, and to the extent that, these other modeling techniques conform to the infrastructure, it will be possible to exchange model elements between the various techniques. For example, a UML model could be input to an entity-relationship model and vice versa. In fact, this is already implemented in some tools.

Features of UML

In addition to the set of diagrams defined in the UML specification, UML provides a set of features that derive from a variety of sources but have all proven valuable in real-world modeling:

✦ **Extensibility mechanisms (stereotypes, tagged values, and constraints):** No standard, language, or tool will ever be able to address 100 percent of the users’ needs. Trying to deliver such a tool would result in a never-ending project without any product. Instead, UML authors focused on a core set of functionality and features. Then they added a set of mechanisms — stereotypes, tagged values, and constraints — that may be used to augment or tailor the core concepts without corrupting the integrity of those core concepts. Applying a stereotype to a model element is like getting dressed up for a special occasion. You remain the same person regardless of what you wear, but the attire helps you fit into the particular situation.

Stereotypes help identify the role of an element within the model without defining or altering its fundamental purpose or function. Stereotypes for model elements work much the same as stereotypes for business descriptions; that is, you might point out a company and identify it as an accounting firm, a shoe distributor, or a grocery store. But it is still a company. Tagged values provide the means to add new model elements that hold values—for example, `author="Tom Pender"`. Constraints allow you to define rules regarding the integrity or use of a model element, such as the attribute "name" must be between 1 and 40 characters including spaces and punctuation, but no special characters. UML added OCL (see Chapter 18) for formally specifying constraints.

✦ **Threads and processes:** Threads and processes are an increasingly common aspect of applications. UML supports the modeling of threads and processes in all of the behavioral models, including the enhanced Activity diagram. (See Chapters 8 through 13.)

✦ **Patterns and collaborations:** In recent years developers have come to appreciate more and more the value of designs based on proven solutions. Patterns and collaborations allow the modelers to define standard approaches to solving common problems. A pattern may then be applied to a variety of specific situations, bringing with it a combination of predefined roles and interactions. Patterns and collaborations may be identified and defined at many levels of abstraction, cataloged, and documented for others to use. This approach brings reuse out of the realm of pure code and into every phase of the modeling effort, from requirements and architecture through implementation.

✦ **Activity diagrams (for business process modeling):** For years business and technical staff have relied on the flowchart. UML renamed the flowchart to Activity diagram. The Activity diagram is a simple yet effective tool to model logic. Logic appears throughout the development process in workflow, method design, screen navigation, calculations, and more. The value of the Activity diagram cannot be overlooked so it has been incorporated into the UML standard since the earliest versions. To bring it up to date, it has been enhanced most recently with its own semantics, distinct from state machines, to represent control flow and/or object flow.

✦ **Refinement (to handle relationships between levels of abstraction):** Many concepts, such as classifiers and relationships, permeate all layers of systems development, and the semantics for these concepts hold true regardless of the business or technical environment. Each abstraction layer adds to, customizes, and otherwise refines the original definition. This approach supports and in some ways encourages the development of varying applications of the concepts at each new level of abstraction. The result of this approach has been the development of an increasingly holistic set of models for systems development, all founded on the same conceptual standard, but each tailored to a unique perspective.

✦ **Interfaces and components:** One advantage of modeling is the ability to work at different levels of abstraction instead of always working at the code level.

Interfaces and components allow the modeler to work on a problem by focusing on the connectivity and communication issues that can help solve that problem. The implementation or even the internal design of a component can be ignored temporarily until the bigger issues of policy, protocol, interface, and communication requirements are resolved. Working at this higher level of abstraction produces a model that later can be, and often is, implemented in multiple environments.

✦ **Constraint language:** The Object Constraint Language (OCL) provides the syntax to define rules that insure the integrity of the model. Much of the constraint concept is borrowed from *programming by contract*, in which relationships between model elements are defined in terms of the rules that govern an interaction. When two parties enter into a contract, the terms of the contract place obligations on the client (the person asking for a product or service) and the supplier (the one providing the product or service). Constraints called *pre-conditions* define what the client must do in order to have the right to receive the product or service. Constraints also define the obligations the supplier must fulfill if the client fulfills her part. These constraints are called *post-conditions* or guarantees. Constraints can also apply to individual elements to define the domain of valid values. (See Chapter 18.)

✦ **Action semantics:** The goal of UML has always been to model software as accurately as possible. Modeling software means modeling behavior. The action semantics extensions enable you to express discrete behaviors as actions. Actions can transform information and/or change the system. Furthermore, UML models actions as individual objects. As such, actions may execute concurrently. In fact, that is their normal mode of execution unless chained together to enforce sequential execution. Settling on concurrent execution as the norm better supports today's distributed environments. Action semantics is also a major contribution toward executable UML. (See Chapter 19.)

Introducing UML 2.0

The next version of UML, 2.0, is due to be released sometime in 2003. Three proposals have been submitted. I have based the content of this book on those submissions and my expectation that they will be adopted in whole or in part. Version 2.0 is a substantial improvement of the underlying architecture, cleaning up many of the fundamental definitions and improving the alignment with the other key technologies sponsored by the OMG.

I've outlined some of the specific objectives for version 2.0 set forth in the RFP. I don't expect beginners to UML to understand them from these very brief descriptions. The rest of this chapter explains many of the new terms. The rest of the book is devoted to explaining how these concepts have been addressed in the diagrams and in the semantics that support the diagrams. For those of you who have been working with UML for a while, these items should demonstrate the OMG's commitment to the long-term success of UML.

Improve the architecture: Rework the physical metamodel so that it is more tightly aligned with the MOF meta-metamodel. Improve the guidelines that establish what constructs should be defined in the kernel language and what constructs should be defined in UML profiles or standard model libraries. (See Chapter 2.)

Provide improved extensibility: Enhance the extensibility mechanisms to align them more closely to a true "four-layer architecture." Profiles provide much of the customization support, at least in concept. But the extensibility features used to create them (stereotypes, tagged values, and constraints) are still rather low-level. UML extensibility features should align more closely with the MOF extensibility features, that is, metaclasses. (See Chapters 2 and 3.)

Improve support for component-based development: Current technologies such as EJB and COM+ require a means to model and manage component-based designs. The current semantics and notation are not quite up to the task. (See Chapters 15 through 17.)

Improve the modeling of relationships: Improve the semantics for `«refinement»` and `«trace»` dependencies. Today it is difficult to support refinement of the models through the life cycle of a project, that is, analysis to design or design to implementation. (See Chapter 6.)

Separate the semantics of statecharts and activity graphs: The initial UML specification tried to define activity graphs as a specialization of a statechart. The overlap has created obstacles to business modeling and has prevented the addition of valuable business modeling features. Support more relaxed concurrency in both diagrams. Support specialization of state machines. (See Chapters 11 and 13.)

Improve model management: Update the notation and semantics for models and subsystems to improve support for enterprise architecture views.

General mechanisms: Define support for model versioning.

The Object Management Group

The organization responsible for developing the UML goals described previously is the Object Management Group (OMG). The OMG is the official steward of the UML standard. This is not simply because the OMG likes standards or likes to take on work, but because it is the driving force behind a much larger plan for software development called Model-Driven Architecture (MDA). MDA is a genuinely ambitious effort to standardize systems development. The goal is to create a complete standard for the creation of implementation-independent models that may be mapped to any platform, present or future. Did I say it was ambitious?

UML plays an integral role in the development and use of the MDA approach. UML is the language used to describe the key standards of MDA, namely UML itself, the Meta-Object Facility (MOF), and the Common Warehouse Model (CWM). UML is also used to create the work products of the MDA process, specifically the business and implementation models.

Model-Driven Architecture (MDA)

Developers usually find that there is a division in most applications between the business logic and the implementation mechanisms to support that logic. For example, selling tickets to a performance is a business practice that could be implemented using any of dozens of technologies and techniques. But no matter how it is implemented, there are fundamental rules that must hold true. If the rules for conducting the business transaction are bound to specific implementation technologies, then changes in those technologies require changes to the rules of the transaction. Such changes incur the risk either of corrupting the transaction or of causing delays while you untangle the business from the technology. This makes even changing the application to take advantage of technological advancements a risk to the business.

Model-Driven Architecture (MDA) separates the two fundamental elements of an application into two distinct models. The platform-independent model (PIM) defines business functionality and behavior, the essence of the system apart from implementation technologies. The platform-specific model (PSM) maps the PIM to a specific technology *without altering the PIM*. That last phrase is critical. Defining a PIM is like defining the job description "bookkeeper". We can define the purpose, responsibilities, qualifications, and skills for the job, that is, the PIM, without knowing who will actually do the job. The PSM corresponds to hiring someone to do the job defined by the bookkeeper job description.

This example highlights the power of the MDA approach. I can hire different people over time. I can even hire multiple people at the same time to perform the bookkeeping duties. I can even take the job description to another company and use it there. In the same manner, I should be able to take the same PIM and deploy it in many technologies or even in different businesses.

The division of the two models also supports interoperability. A business function does not need to know the implementation of another business function in order to access it. The interface is defined in PIM fashion. The PSM takes care of the mapping to the implementation. So the calling function is unaffected by changes to the implementation of the called function. In the bookkeeper example, the bookkeeper PIM/job description can define an interface to the general ledger. Where or how the general ledger is implemented is irrelevant. The interface is always the same. Whether I implement the bookkeeping system at a department store, software consulting firm, or insurance company, and whether the system is implemented in .NET or Java One, the interaction between the bookkeeper and the general ledger is the same. So, as technologies change over time, as they inevitably will, the business remains stable and relatively unaffected by the changes.

This same concept applies to functions that should be globally available to systems such as transaction management, domain specific services, and application services. Having learned from its experience with the CORBA-based Object Management Architecture, OMG recognizes the need for three levels of MDA-based specifications built on the standardized technologies already defined by the OMG (see Figure 1-1):

✦ **Pervasive Services** include security, transaction management, directory support, and event generation and handling common to most systems.

✦ **Domain Facilities** include standardized models for subject areas such as telecom, space sciences, biotechnology, and finance.

✦ **Applications** are within a domain, such as a heart monitor in the biotechnology domain or a funds transfer application in a financial domain.

The core of the MDA is the set of standards (MOF, UML, CWM, and XMI) and technologies (CORBA, .NET, Java, and so on). The pervasive services are built on that core. Then based on these standards and services, businesses can build domain specific profiles for finance, e-commerce and so on. Within each domain, businesses may then build specific applications that conform to the supporting standards.

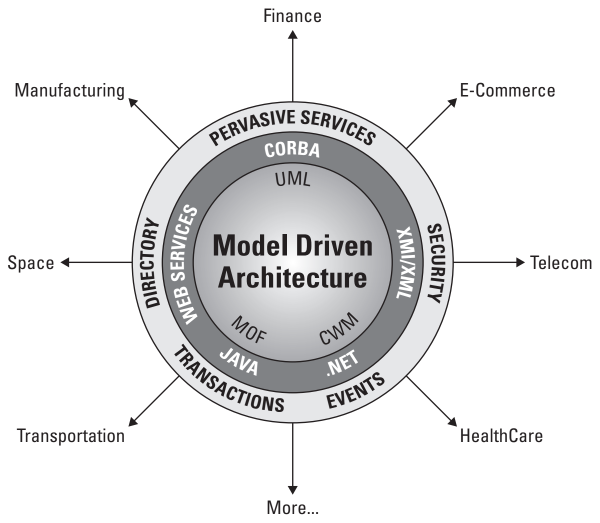

Figure 1-1: The Model-Driven Architecture.
©OMG

The Pervasive Services

The features included in the Pervasive Services level are those commonly found in the existing list of CORBA services:

✦ Directory services

✦ Transactions

✦ Event handling/notification

✦ Security

The list is sure to grow in time with additions from the OMG itself, based on CORBA, and from OMG members. Work has already started on mapping these services to PIMs so that they can be applied to all platforms through the MDA development approach.

Domain Facilities

A domain is simply a subject area, such as warehousing and distribution, or biotechnology. Each domain has its own peculiar set of problems and concepts. For example, there are many banks but they all conduct the same type of business. The resources they use, the behaviors they support, and even many of the regulations that govern their performance are the same. The differences arise in how they choose to embellish the basic business to appeal to their customers and to improve profitability. A *domain* is the description of the fundamental elements common to all systems in the same subject area. The uses of those fundamental elements define the applications within the domain. These applications are covered in the next section.

Work has already begun on a number of domain models. Even though MDA-based standards for specific domains are still under development, OMG Domain Task Forces (DTF) have started to apply MDA to existing projects. For example, OMG's Life Science Research DTF, working in biotechnology, has already modified its Mission and Goals Statement to reflect its work in MDA. In mid-2000, even before MDA, OMG's Healthcare DTF (formerly know by its nickname, CORBAmed) published its Clinical Image Access Service (CIAS) (`www.omg.org/cgi-bin/doc?dtc/01-07-01`) including a nonstandard UML model that describes the specification written in OMG IDL. The document provides a good example of what a future MDA specification might look like.

Note

In a true MDA specification, the model follows the UML standard and is fully developed, defining all interfaces and operations including parameters and types, and specifying pre- and post-conditions in OCL.

MDA Success Stories

Companies who have applied/are applying MDA include:

- ✦ Regions Bank of Birmingham, Alabama
- ✦ Swedish Parliament
- ✦ Deutsche Bank Bauspar AG
- ✦ U.S. Government Intelligence Agency
- ✦ The Open System Architecture for Condition Based Monitoring (OSA-CBM) Project
- ✦ CGI
- ✦ ff-eCommerce
- ✦ Swisslog Software AG
- ✦ Adaptive; Adaptive Framework
- ✦ Financial Systems Architects
- ✦ Headway Software; Headway review
- ✦ IKV++ GmbH; m2c(tm)

Applications

For years, businesses have started projects by modeling the business application requirements. As the projects proceeded, they fell deeper and deeper into implementation-dependent modeling, often losing sight of the original business requirements in the midst of the overwhelming task of working with ever-changing implementation technologies. As MDA-based development tools become more widely available, projects can be focused more on the platform independent model of the business requirements. In fact, the focus throughout the project will remain on the original requirements while the implementation becomes more and more automated through the application of platform specific models.

Lest you think that this is a pipe dream, take a look at the list of companies in the sidebar who are already using this technique successfully. Many more companies are listed at `www.omg.org/mda/products_success.htm`.

Meta-Object Facility (MOF)

The Meta-Object Facility (MOF) is at the heart of the MDA strategy along with the UML, CWM, CORBA, and XMI. It is the starting point, the standard that defines the languages used to describe systems and MDA itself. The MOF is a *metamodel* (often called *M2*), a model defining the concepts required to build a model and to store the model in a repository. The model is stored by representing the metadata as CORBA objects.

Cross-Reference

Models, metamodels, and meta-metamodels are more fully explained in Chapter 2.

Currently the MOF defines all the foundation concepts needed to build the two modeling languages UML and CWM. Now just to make this a little more confusing, both UML and CWM are themselves metamodels. They are models that define modeling languages. When a metamodel like MOF is used to define another metamodel, it becomes a *meta-metamodel*, or *M3* for short. Since all elements defined by UML or CWM conform to the MOF standard, it is possible to define a standardized repository for all data generated in UML or CWM or, in the future, any other languages derived from MOF.

The model elements in the UML are created, or instantiated, from model elements defined in the MOF. For example, the MOF defines the concept "Classifier." UML defines a concept called "Classifier" that inherits the description in the MOF and then adds to it for the purpose of modeling objects. CWM also inherits "Classifier" but for a different reason: CWM adds to the "Classifier" definition to support modeling data. Figure 1-2 illustrates this relationship between the three models.

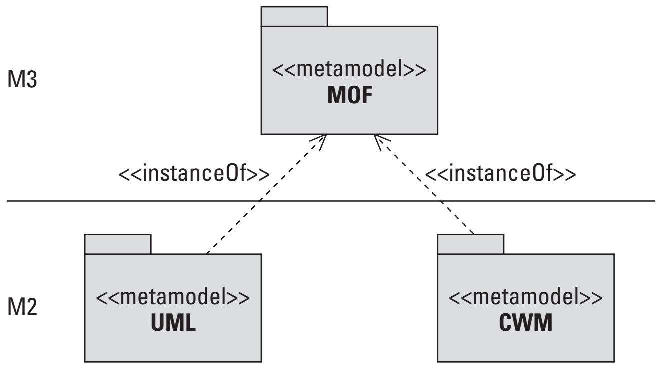

Figure 1-2: The relationship between the MOF and the UML and CWM languages.
©OMG 2.0

MOF is also part of the long-term OMG strategy to support the creation and exchange of a variety of metamodels across diverse repositories. For example, using the MOF, a UML model might be transmitted between tools by different vendors. Likewise, a UML object model might be ported to a data-modeling tool in order to derive a logical data model from the object model.

MOF supports this long-term strategy by providing

✦ The infrastructure for implementing CORBA-based design and reuse repositories

✦ The definition for a set of CORBA IDL interfaces to define and manipulate metamodels and the models created using them

✦ The rules for automatically generating the CORBA interfaces for metamodels, thus insuring consistency

Common Warehouse Metamodel (CWM)

The Common Warehouse Model (CWM) was developed in cooperation with the Meta-Data Coalition (MDC). The goal of CWM was to provide to the data modeling community the same type of solution that UML provided to the object modeling community. In the same way that UML describes a common modeling language for building systems, CWM describes metadata interchange among data warehousing, business intelligence, knowledge management, and portal technologies. Like UML, CWM is a language derived from the MOF. CWM provides the mapping from MDA PIMs to database schemas. CWM covers the full life cycle of designing, building, and managing data warehouse applications and supports management of the life cycle.

You can find the specifications for CWM at `www.omg.org/technology/documents/formal/cwm.htm`. Two other specifications to extend CWM to the Internet are also currently under way: CWM Web Services (`www.omg.org/techprocess/meetings/schedule/CWM_Web_Services_RFP.html`) and CWM Metadata Interchange Patterns (MIP) (`www.omg.org/techprocess/meetings/schedule/CWM_MIP_RFP.html`).

XML Metadata Interchange (XMI)

At its simplest level, XMI defines a mapping from UML to XML. It defines standard formats and Document Type Definitions (DTD) to capture UML models (and metamodels). This makes it possible to then convert a UML model into XML, distribute it pretty much anywhere, and then convert it back to UML. The mapping also makes it possible to exchange UML models between tools and across platforms.

Cross-Reference

You can read more about XMI in Chapter 22.

Technically, XMI mapping uses MOF metadata, not UML. But since UML is based on the MOF metadata, anything defined by UML is compatible with XMI mapping features. Additional work is being done to extend XMI to support W3C-standard XML schema.

Summary

UML grew out of the increasingly complex challenge to build systems that not only met users' requirements but that could withstand the ever-changing technological environment. Change, complexity, and speed conspired to focus critical attention on how to build robust, durable systems. One result was a standard language for modeling systems, the Unified Modeling Language (UML).

But the desire for truly industry-capable tools to build systems did not stop there. The OMG has continued to spearhead the effort to build a comprehensive strategy in the form of Model-Driven Architecture (MDA).

There are a lot of languages involved in these strategies. Here's how they all relate:

✦ The Meta Object Facility (MOF) defines a common meta-language for building other languages.

✦ UML defines a meta-language, derived from the MOF, for describing object-oriented systems.

✦ The Common Warehouse Metamodel defines a meta-language, derived from the MOF, for describing data warehousing and related systems.

✦ XML Metadata Interchange defines the means to share models derived from the MOF.

✦ ✦ ✦